# Conflict Equivalence of Branching Processes

David Delfieu, Maurice Comlan
Polytech'Nantes
Institute of Research on Communications
and Cybernetics of Nantes, France
Email: david.delfieu@irccyn.ec-nantes.fr
maurice.comlan@irccyn.ec-nantes.fr
Médésu Sogbohossou
Polytechnic School of Abomey-Calavi
Laboratory of Electronics, Telecommunications
and Applied Computer Science, Abomey-Calavi, Benin
Email: sogbohossou_medesu@yahoo.fr

***Abstract*—For concurrent and large systems, specification step is a crucial point. Combinatory explosion is a limit that can be encountered when a state space exploration is driven on large specification modeled with Petri nets. Considering bounded Petri nets, technics like unfolding can be a way to cope with this problem. This paper is a first attempt to present an axiomatic model to produce the set of processes of unfoldings into a canonic form. This canonic form allows to define a conflict equivalence.**

***Index Terms*—Petri Nets; Unfolding; Branching process; Algebra.**

## I. Introduction

The complexity and the criticity of some real-time system (transportation systems, robotics), but also the fact that we can no longer tolerate failures in less critical realtime systems (smartphones, warning radar devices) enforces the use of verification and validation methods. Petri nets are a widely used tool used to model critical real-time systems. The formal validation of properties is then based on the computation of state space. But, this computation faces generally, for highly concurrent and large systems, to combinatory explosion.

The specification of parallel components is generally modeled by the interleavings of the behavior of each components. This semantics of interleaving is exponentially costly in the computing of the state space. Partial order semantics have been introduced to shunt those interleavings. This semantics prevents combinatory explosion by keeping parallelism in the model.

The objective of this approach is to pursue a theoretical aspect: to speed up the identification of the branching processes of an unfolding. The notion of equivalence can be used to make a new type of reduction of unfoldings.

Finite prefixes of net unfoldings constitute a first transformation of the initial Petri Net (PN), where cycles have been flattened. This computation produces a process set where conflicts act as a discriminating factor. A conflict partitions a process in branching processes. An unfolding can be transformed into a set of finite branching processes. These processes constitute a set of acyclic graphs - several graphs can be produced when the PN contains parallelism - built with events and conditions, and structured with two operators: causality and true parallelism. An interesting particularity of an unfolding is that, in spite of the loss of global marking, these processes contain enough information to reconstitute the reachable markings of the original Petri nets. In most of the cases, unfoldings are larger than the original Petri net. This is provoked essentially when values of precondition places exceed the precondition of non simple conflicts. This produces a lot of alternative conditions. In spite of that, a step has been taken forward: cycles have been broken and the conflicts have structured the nets in branching processes.

This paper proposes proposes an algebraic model for the definition and the reduction of the branching process of an unfolding. This paper extends [1] to reset Petri nets. Reset arcs are particularly useful, they bring expressiveness and compactness. In the example presented in the Section VI, reset arcs allow to clear the states particularly when the user has several attempts to enter its code.

A lot of works have been proposed to improve unfolding algorithms [2][3][4][5]. Is there another way to draw on recent works about unfolding? In spite of the eventual increase of the size of the net unfoldings, the suppression of conflicts and loops has decreased its structural complexity, allowing to compute the state space and to the extract of semantic information.

From a developer's point of view an unfolding can be efficiently coded by a boolean table of events. This table describes every pair to pair relation between events. This table has been the starting point of our reflection: it stresses the point that a new connector can be defined to express that a set of events belong to the same process. This connector allows to aggregate all the events of a branching process. For example, a theorem is proposed to compute all the branching processes, in canonic form, for chains of conflicts of the kind illustrated in Figure 1.

The work presented in this paper takes place in the context

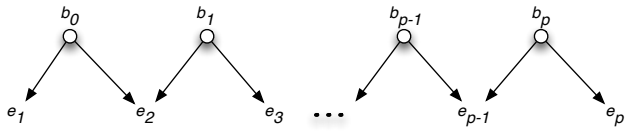


Figure 1. Chain of conflicts.

TABLE I. Process syntax.

| | | | |
|---|---|---|---|
| $Capacity$ | $\alpha$ | := | $\bar{x} \mid x \mid \tau$ |
| $Proces$ | $p$ | ::= | $\alpha.p \mid p \mid\mid q \mid p+q \mid D(\tilde{x}) \mid p\backslash x \mid 0$ |

of combining process algebra [6][7] and Petri nets [8].

The axiomatic model of Milner's process with Calculus of Communicating Systems (CCS) is compared with the branching processes and related to other works in Section II. Then, after a brief presentation of Petri nets and unfoldings in Section III, Section IV presents our contribution with the definition of an axiomatic framework and the description of properties. The last section presents examples, in particular, illustrating a conflict equivalence.

## II. Related Work

Process algebra appeared with Milner [7] on the Calculus of Communicating Systems (CCS) and the Communicating Sequential Processes (CSP) of Hoare [6]. These approaches are not equivalent but share similar objectives. The algebra of branching process proposed in this paper is inspired by the process algebra of Milner. $CCS$ is based on two central ideas: The notion of observability and the concept of synchronized communication; $CCS$ is as an abstract code that corresponds to a real program whose primitives are reduced to simple send and receive on channels. The terms (or agents) are called processes with interaction capabilities that match requests communication channels. The elements of the alphabet are observable events and concurrent systems (processes). They can be specified with the use of three operators: sequence, choice, and parallelism. A main axiom of $CCS$ is the rejection of distributivity of the sequence upon the choice. Let $p$ and $q$ be two processes, the complete syntax of process is described in the Table I.

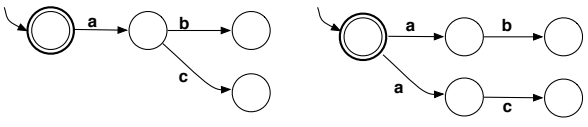


Figure 2. Milner: rejection of distributivity of sequence on choice.

Consider an observer. In the left automaton of Figure 2, after the occurrence of the action $a$, he can observe either $b$ or $c$. In right automaton, the observation of $a$ does not imply that $b$ and $c$ stay observable. The behavior of the two automata are not equivalent.

In $CCS$, Milner defines the observational equivalence. Two automata are observational equivalent if there are bisimular. On a algebraic point of view, the distributivity of the sequence on the choice is rejected in equation (1):

$$a.(b+c) \not\equiv_{behaviorally} a.b + a.c \tag{1}$$

The key point of our approach is based on the fact that this distributivity is not rejected in occurrence nets. The timing of the choices in a process is essential [9]. The nodes of occurrence nets are events. An event is a fired transition of the underlying Petri net. In $CCS$, an observer observes possible futures. In occurrence nets, the observer observes arborescent past. This controversy in the theory of concurrency is an important topic of linear time versus branching time. In the model, equation (2) holds:

$$a \prec (b \perp c) \equiv (a \prec b) \perp (a \prec c) \tag{2}$$

Equation (2) is a basic axiom of our algebraic model. The equivalence relation differs then from bisimulation equivalence. This relation will be defined in the following with the definition of the canonic form of an unfolding.

Branching process does not fit with process algebra on numerous other aspects. For example, a difference can be noticed about parallelism. While unfolding keeps true parallelism, process algebra considers a parallelism of interleaving. Another difference is relative to events and conditions, which are nodes of different nature in an unfolding. Conditions and events differ in term of ancestor. Every condition is produced by at most one event ancestor (none for the condition standing for $m_0$, the initial marking), whereas every event may have $1$ or $n$ condition ancestor(s).

In $CCS$, there is no distinction between conditions and events. Moreover, conditions will be consumed defining processes as set of events. However, a lot of works [5][9][10] have shown the interest of an algebraic formalization: it allows the study of connectives, the compositionally and facilitates reasoning (tools like [11]). Let have two Petri nets; it is questionable whether they are equivalent. In principle, they are equivalent if they are executed strictly in the same manner. This is obviously a too restrictive view they may have the same capabilities of interaction without having the same internal implementations. These works resulted to find matches (rather flexible and not strict) between nets. Mention may be made among other the occurrence net equivalence [12], the bisimulation equivalence [13], the partial order equivalence [14], or the ST-bisimulation equivalence [15]. These different equivalences are based either on the isomorphism between the unfolding of nets or on observable actions or traces of the execution of Petri nets or other criteria.

The approach developed in this paper proposes a new equivalence, which is weaker than a trace equivalence; it does not preserves traces but preserves conflicts. The originality of the approach is to encapsulate causality and concurrency in a new operator, which "aggregates" and "abstracts" events in a process. This new operator reduces the representation and accelerates the reduction process. This paper intends first, to give an algebraic model to an unfolding, and second,

to establish a canonic form leading to the definition of an equivalence conflict.

## III. Unfolding a Petri Net

In this section, Petri nets and unfolding of Petri nets are presented.

### A. Petri Net

A Petri net [8] $\mathcal{N} =< P, T, \mathcal{W} >$ is a triple with: $P$, a finite set of places, $T$, the finite set of transitions, $P \cup T$ are nodes of the net; ($P \cap T = \emptyset$ signifies that $P$ and $T$ are disjoint), and $\mathcal{W} : (P \times T) \cup (T \times P) \to \mathcal{N}$, the flow relation defining arcs (and their valuations) between nodes of $\mathcal{N}$. A *marking* of $\mathcal{N}$ is a multiset M: $P \to \{0, 1, 2, ...\}$ and the *initial marking* is denoted $M_0$.

The pre-set (resp. post-set) of a node $x$ is denoted ${}^\bullet x = \{y \in P \cup T \mid \mathcal{W}(y, x) > 0\}$ (resp. $x^\bullet = \{y \in P \cup T \mid \mathcal{W}(x, y) > 0\}$). A transition $t \in T$ is said *enabled* by $m$ iff: $\forall p \in {}^\bullet t,\ m(p) \geq \mathcal{W}(p, t)$. This is denoted: $m \xrightarrow{t}$ Firing of $t$ leads to the new marking $m'$ ($m \xrightarrow{t} m'$): $\forall p \in P,\ m'(p) = m(p) - \mathcal{W}(p, t) + \mathcal{W}(t, p)$. The initial marking is denoted $m_0$.

A Petri net is *k-bounded* iff $\forall m$, reachable from $m_0, m(p) \leq k$ (with $p \in P$). It is said *safe* when *1-bounded*. Two transitions are in a *structural conflict* when they share at least one pre-set place; a conflict is *effective* when these transitions are both enabled by a same marking. The considered Petri nets in this paper are k-bounded.

Reset arcs constitute an extension of Petri nets. These arcs does not change the enabling rules of transitions [16]. If $Rst(p, t)$ represents the set of reset arcs from a transition $t$ to a place $p$. If $M \xrightarrow{t} M'$ then $\forall p \in P$ such as $Rst(p, t) = 0$, $M'(p) = 0$. But if $W(t, p) > 0$ then $M'(p) = W(t, p)$. The firing rule is defined by the following relation

$$\forall p \in P, \quad M' = (M - Pre(p, t)) \ . \ R(p, t) + Post(p, t)$$

where "." is the Hadamard matrix product.

**Definition 1** (Reset arc Petri Nets)**.** *A reset arc Petri Nets is a tuple $N_R =< P, T, W, R >$ with $< P, T, W >$ a Petri nets and $Rst : P \times T \to \{0, 1\}$ is the set of reset arcs ($Rst(p, t) = 0$ is there exists a reset arc binding p to t, else $Rst(p, t) = 1$).*

### B. Unfolding

In [3], the notion of *branching process* is defined as an initial part of a run of a Petri net respecting its partial order semantics and possibly including non deterministic choices (conflicts). This net is acyclic and the largest branching process of an initially marked Petri net is called *the* unfolding of this net. Resulting net from an unfolding is a labeled occurrence net, a Petri net whose places are called *conditions* (labeled with their corresponding place name in the original net) and transitions are called *events* (labeled with their corresponding transition name in the original net).

An occurrence net [17] is a net $\mathcal{O} =< \mathcal{B}, \mathcal{E}, \mathcal{F} >$, where $\mathcal{B}$ is the set of *conditions* (places), $\mathcal{E}$ is the set of *events* (transitions), and $\mathcal{F}$ the flow relation (1-valued arcs), such that:

- for every b $\in \mathcal{B}$, $|{}^\bullet b| \leq 1$;
- $\mathcal{O}$ is acyclic;
- for every e $\in \mathcal{E}$, ${}^\bullet e \neq \emptyset$;
- $\mathcal{O}$ is finited preceded;
- no element of $\mathcal{B} \cup \mathcal{E}$ is in conflict with itself;
- $\mathcal{F}^+$, the transitive closure of $\mathcal{F}$, is a strict order relation.

$Min(\mathcal{O}) = \{b \mid b \in \mathcal{B}, |{}^\bullet b| = 0\}$ is the minimal conditions set: the set of conditions with no ancestor can be mapped with the initial marking of the underlying Petri net. Also, $Max(\mathcal{O}) = \{x \mid x \in \mathcal{B} \cup \mathcal{E}, |x^\bullet| = 0\}$ are maximal nodes. A *configuration C* of an occurence net is a set of events satisfying:

- if $e \in C$ then $\forall e' \prec e$ implies $e' \in C$ ($C$ is causally closed);
- $\forall e, e' \in C : \neg(e \perp e')$ ($C$ is conflict-free).

A *local configuration* [e] of an event e is the set of event e', such that $e' \prec e$.

Three kinds of relations could be defined between the nodes of $\mathcal{O}$:

- The strict causality relation noted $\prec$: for $x, y \in \mathcal{B} \cup \mathcal{E},\ x \prec y$ if $(x, y) \in \mathcal{F}^+$ (for example $e_3 \prec e_6$, in Figure 3.b).
- The conflict relation noted $\perp$: $\forall b \in \mathcal{B}$, if $e_1, e_2 \in b^\bullet$ ($e_1 \neq e_2$), then $e_1$ and $e_2$ are in *conflict relation*, denoted $e_1 \ \perp \ e_2$ (for example $e_4 \perp e_5$, in Figure 3.b).
- The concurrency relation noted $\wr$: $\forall x, y \in \mathcal{B} \cup \mathcal{E}$ ($x \neq y$), $x \wr y$ ssi $\neg((x \prec y) \vee (y \prec x) \vee (x \ \perp \ y))$ (for example $e_2 \wr e_3$, in Figure 3.b).

**Remark 1.** *The transitive aspect of $\mathcal{F}^+$ implies a transitive definition of strict causality.*

A set $B \subseteq \mathcal{B}$ of conditions such as $\forall b, b' \in B, b \neq b' \Rightarrow b \wr b'$ is a *cut*. Let $B$ be a *cut* with $\forall b \in B, \nexists b' \in \mathcal{B} \backslash B,\ b \wr b'$, $B$ is the *maximal cut*.

**Definition 2.** The unfolding $Unf_F \stackrel{\text{def}}{=} < \mathcal{O}_F, \lambda_F >$ *of a marked net $< \mathcal{N}, m_0 >$, with $\mathcal{O}_F \stackrel{\text{def}}{=} < \mathcal{B}_F, \mathcal{E}_F, \mathcal{F}_F >$ an occurrence net and $\lambda_F : \mathcal{B}_F \cup \mathcal{E}_F \to \mathcal{P} \cup \mathcal{T}$ (such as $\lambda(\mathcal{B}_F) \subseteq \mathcal{P}$ and $\lambda(\mathcal{E}_F) \subseteq \mathcal{T}$) a labeling function, is given by:*

1) *$\forall p \in \mathcal{P}$, if $m_0(p) \neq \emptyset$, then $B_p \stackrel{\text{def}}{=} \{b \in \mathcal{B}_F \mid \lambda_F(b) = p \wedge {}^\bullet b = \emptyset\}$ and $m_0(p) = |B_p|$;*
2) *$\forall B_t \subseteq \mathcal{B}_F$ such as $B_t$ is a cut, if $\exists t \in \mathcal{T}, \lambda_F(B_t) = {}^\bullet t \wedge |B_t| = |{}^\bullet t|$, then:*
   a) *$\exists! e \in \mathcal{E}_F$ such as ${}^\bullet e = B_t \wedge \lambda_F(e) = t$;*
   b) *if $t^\bullet \neq \emptyset$, then $B'_t \stackrel{\text{def}}{=} \{b \in \mathcal{B}_F \mid {}^\bullet b = \{e\}\}$ is as $\lambda_F(B'_t) = t^\bullet \wedge |B'_t| = |t^\bullet|$;*
   c) *if $t^\bullet = \emptyset$, then $B'_t \stackrel{\text{def}}{=} \{b \in \mathcal{B}_F \mid {}^\bullet b = \{e\}\}$ is as $\lambda_F(B'_t) = \emptyset \wedge |B'_t| = 1$;*

3) $\forall B_t \subseteq \mathcal{B}_F$, *if* $B_t$ *is not a cut , then* $\nexists e \in \mathcal{E}_F$ *such as* ${}^\bullet e = B_t$.

Definition 2 represents an *exhaustive* unfolding algorithm of $< \mathcal{N}, m_0 >$. In 1., the algorithm for the building of the unfolding starts with the creation of conditions corresponding to the initial marking of $< \mathcal{N}, m_0 >$ and in 2., new events are added one at a time together with their output conditions (taking into account sink transitions). In 3., the algorithm requires that any event is a possible action: there are no adding nodes to those created in items 1 and 2. The algorithm does not necessary terminate; it terminates if and only if the net $< \mathcal{N}, m_0 >$ does not have any infinite sequence. The sink transitions (ie $t \in \mathcal{T}, t^\bullet = \emptyset$) are taken into account in 2.(c).

Let be $\mathcal{E} \subset \mathcal{E}_F$. The occurrence net $\mathcal{O} \stackrel{\text{def}}{=} < \mathcal{B}, \mathcal{E}, \mathcal{F} >$ associated with $\mathcal{E}$ such as $\mathcal{B} \stackrel{\text{def}}{=} \{b \in \mathcal{B}_F \mid \exists e \in \mathcal{E}, b \in {}^\bullet e \cup e^\bullet\}$ and $\mathcal{F} \stackrel{\text{def}}{=} \{(x,y) \in \mathcal{F}_F \mid x \in \mathcal{E} \vee y \in \mathcal{E}\}$ is *a prefix* of $\mathcal{O}_F$ if $Min(\mathcal{O}) = Min(\mathcal{O}_F)$. By extension, $Unf \stackrel{\text{def}}{=} < \mathcal{O}, \lambda >$ (with $\lambda$, the restriction of $\lambda_F$ to $\mathcal{B} \cup \mathcal{E}$) is a prefix of unfolding $Unf_F$.

It should be noted that, according to the implementation, the names (the elements in the sets $\mathcal{E}$ and $\mathcal{B}$) given to nodes in the same unfolding can be different. A name can be independently chosen in an implementation using a tree formed by its causal predecessors and the name of the corresponding nodes in $\mathcal{N}$ [3].

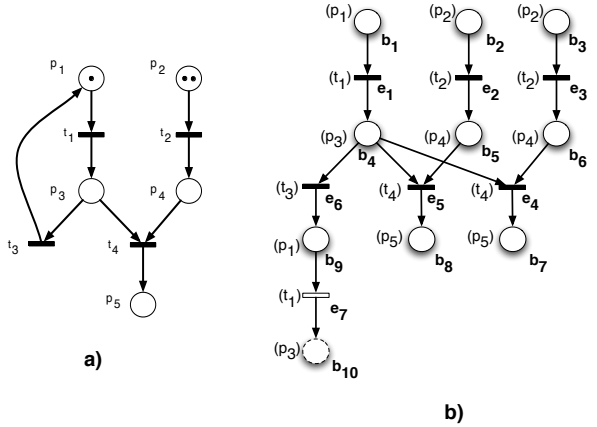


Figure 3. a) Petri net, b) Unfolding.

**Definition 3.** *A causal net* $\mathcal{C}$ *is an occurrence net* $\mathcal{C} \stackrel{\text{def}}{=} < \mathcal{B}, \mathcal{E}, \mathcal{F} >$ *such as:*

1) $\forall e \in \mathcal{E} : e^\bullet \neq \emptyset \ \wedge \ {}^\bullet e \neq \emptyset$;
2) $\forall b \in \mathcal{B} : |b^\bullet| \leq 1 \ \wedge \ |{}^\bullet b| \leq 1$.

**Definition 4.** $\mathcal{P}_i = (\mathcal{C}_i, \lambda_F)$ *is a* process *of* $< \mathcal{N}, m_0 >$ *iff:* $\mathcal{C}_i \stackrel{\text{def}}{=} < \mathcal{B}_i, \mathcal{E}_i, \mathcal{F}_i >$ *is a causal net and* $\lambda : \mathcal{B}_i \cup \mathcal{E}_i \rightarrow P \cup T$ *is a labeling fonction such as:*

1) $\mathcal{B}_i \subseteq \mathcal{B}_F$ *and* $\mathcal{E}_i \subseteq \mathcal{E}_F$
2) $\lambda_F(\mathcal{B}_i) \subseteq P$ *and* $\lambda_F(\mathcal{E}_i) \subseteq T$;
3) $\lambda_F({}^\bullet e) = {}^\bullet \lambda_F(e)$ *and* $\lambda_F(e^\bullet) = \lambda_F(e)^\bullet$
4) $\forall e_i \in \mathcal{E}_i, \ \forall p \in P : \ \mathcal{W}(p, \lambda_F(e)) = |\lambda^{-1}(p) \cap {}^\bullet e| \ \wedge \ \mathcal{W}(\lambda_F(e), p) = |\lambda^{-1}(p) \cap e^\bullet|$
5) *If* $p \in Min(P) \Rightarrow \exists b \in \mathcal{B}_i : \ {}^\bullet b = \emptyset \ \wedge \ \lambda_F(b) = p$

$Max(\mathcal{C}_i)$ is the *state* of $\mathcal{N}$. $Min(\mathcal{C}_i)$ and $Max(\mathcal{C}_i)$ are (resp. minimum) maximum cuts. Generally, any maximal cut $B \subseteq \mathcal{B}_i$ corresponds to a reachable marking $m$ of $< \mathcal{N}, m_0 >$ such as $\forall p \in \mathcal{P}, m(p) = |B_p|$ avec $B_p = \{b \in B \mid \lambda(b) = p\}$.

The *local configuration* of an event $e$ is defined by: $[e] \stackrel{\text{def}}{=} \{e' \mid e' \prec e\} \cup \{e\}$ and is a process. For example of unfolding in Figure 3.b: $[e_4] \stackrel{\text{def}}{=} \{e_1, e_3, e_4\}$.

The conflicts in an unfolding derive from the fact that there is a reachable marking (a cut in an unfolding) such as two or many transitions of a labelled net $< \mathcal{N}, m_0 >$ are *enabled* and the firing of one transition disable other. Whence the proposition:

**Proposition 1.** *Let be* $e_1, e_2 \in \mathcal{E}_F$. *If* $e_1 \ \perp \ e_2$, *then there* $\exists (e'_1, e'_2) \in [e_1] \times [e_2]$ *such as* ${}^\bullet e'_1 \cap {}^\bullet e'_2 \neq \emptyset$ *et* ${}^\bullet e'_1 \cup {}^\bullet e'_2$ *is a cut.*

## IV. Branching Process Algebra

The Section III-B showed how unfolding exhibits causal nets and conflicts. Otherwise, every couple of events that are not bounded by a causal relation or the same conflict set are in concurrency. Then, an unfolding allows to build a $2D$-table making explicit every binary relations between events. Practically, this table establishes the relations of causality and exclusion. If a binary relation is not explicit in the table, it means that the couple of events are in a concurrency relation.

Let $\mathcal{EB} = \mathcal{E} \cup \mathcal{B}$ a finite alphabet, composed of the events and the conditions generated by the unfolding. The event table (produced by the unfolding) defines for every couple in $\mathcal{EB}$ either a causality relation $\mathcal{C}$, either a concurrency relation $\mathcal{I}$ or an exclusive relation $\mathcal{X}$. These sets of binary relations dot not intersect and the following expressions can be deduced:

$$Unf/_{\mathcal{X}} = \mathcal{C} \cup \mathcal{I} \tag{3}$$
$$Unf/_{\mathcal{C}} = \mathcal{X} \cup \mathcal{I} \tag{4}$$
$$Unf/_{\mathcal{I}} = \mathcal{C} \cup \mathcal{X} \tag{5}$$

To illustrate these relation sets, the negation operator noted $\neg$ can be introduced. Then, equations (3), (4), (5) lead to (6), (7), (8):

$$\neg((e_1, e_2) \in \mathcal{I}) \ \Leftrightarrow \ (e_1, e_2) \in \mathcal{C} \cup \mathcal{X} \tag{6}$$
$$\neg((e_1, e_2) \in \mathcal{C}) \ \Leftrightarrow \ (e_1, e_2) \in \mathcal{I} \cup \mathcal{X} \tag{7}$$
$$\neg((e_1, e_2) \in \mathcal{X}) \ \Leftrightarrow \ (e_1, e_2) \in \mathcal{C} \cup \mathcal{I} \tag{8}$$

Equation (8) expresses that if two events are not in conflict they are in the same branching process. Let us now define the union of binary relations $\mathcal{C}$ and $\mathcal{I}$: $\mathcal{P} = \mathcal{C} \cup \mathcal{I}$. For every couple $(e_1, e_2) \in \mathcal{P}$, either $(e_1, e_2)$ are in causality or in concurrency: $\mathcal{P}$ is the union of every branching process of an unfolding.

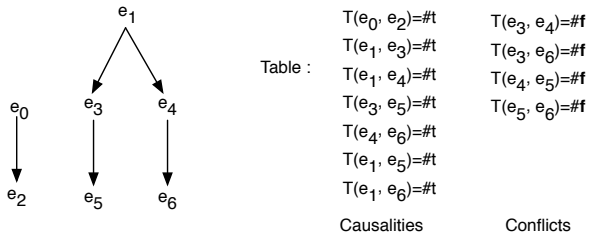


Figure 4. Unfolding.

*a) Example:* Figure 4 represents an unfolding (in the left part) and a Table $T$ (right part), which defines the event relations of the unfolding.

In Figure 4, the Table $T$ contains 7 causal relations and 4 conflict relations. $(e_0, e_4)$ is not (negation) in the table, it means that $e_0$ and $e_4$ are concurrent. Moreover, if two events are not in conflict (consider $e_0$ and $e_6$): $(e_0, e_6)$ is not a key of the table, $(e_0, e_6)$ are in concurrency and thus, those events belongs to the same branching process.

### A. Definition of the Algebra

The starting point of this work is based on the fact that the logical negation operator articulates the relation between two sets: the process set $\mathcal{P}$ and the exclusion set $\mathcal{X}$. As mentioned in Section IV, $\mathcal{C}$, $\mathcal{I}$ *and* $\mathcal{P}$ does not intersect, then semantically, if a couple of events is not in a relation of exclusion (noted $\perp$), the events are in $\mathcal{P}$. $\mathcal{P}$ contains binary relations between events that are in branching process.

To express that events are in the same branching process, a new operator noted $\oplus$ is introduced. An algebra describing branching process can be defined as follow:

$$\{\mathcal{U}, \prec\ ,\ \wr\ ,\ \perp\ , \oplus,\ \neg\}$$

Let us note; $* = \oplus, \prec,$ or $\perp$, $\#t$ the void process, and $\#f$ the false process. Here is the formal signature of the language:

- $\forall e \in \mathcal{EB}, e \in \mathcal{U}, \#t \in \mathcal{U}, \#f \in \mathcal{U}$
- $\forall e \in \mathcal{U}, \neg e \in \mathcal{U}$
- $\forall (e_1, e_2) \in \mathcal{U}^2, e_1 * e_2 \in \mathcal{U}$.

### B. Definition of operators

*1) Causality:* $\mathcal{C}$ is the set of all the causalities between every elements of $\mathcal{EB}$. $e_1 \prec e_2$ if $e_1$ is in the local configuration of $e_2$, i.e., the Petri net contains a path with at least one arc leading from $e_1$ to $e_2$:

$$e_1 \prec e_2 \text{ if } e_1 \in [e_2] \tag{9}$$

- $\prec$ is associative: $e_1 \prec (e_3 \prec e_5) \equiv (e_1 \prec e_3) \prec e_5$;
- $\prec$ is transitive: $(e_1 \prec e_3) \vee (e_3 \prec e_5) \equiv e_1 \prec e_5$;
- $\prec$ is not commutative: $e_1 \prec e_3$ but $e_3 \neg \prec e_1$;
- $\#t$ is the neutral element for $\prec$: $\#t \prec e \equiv e$;
- every element of $\mathcal{EB}$ has an opposite: $\#f \prec e \equiv \neg e$.

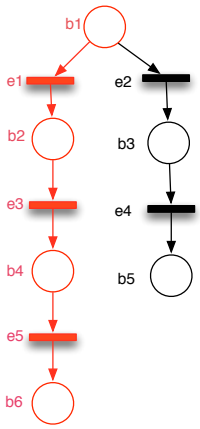


Figure 5. Causalite.

*2) Exclusion:* $\mathcal{X}$ is the set of all the exclusion relations between every elements of $\mathcal{EB}$. Two events $e$ and $e'$ are in exclusion if the net contains two paths $b\ e_1\ ...\ e$ and $b\ e_2\ ...\ e'$ starting at the same condition $b$ and $e_1 \neq e_2$:

$$e_1 \perp e_2 \equiv ((\bullet e_1 \cap \bullet e_2 \neq \emptyset) \text{ or } (\exists e_i, e_i \prec e_2 \text{ and } e_1 \perp e_i)) \tag{10}$$

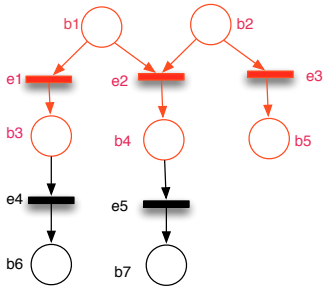


Figure 6. Exclusion.

- $\perp$ is commutative: $e_1 \perp e_2 \equiv e_2 \perp e_1$;
- $\perp$ is associative: $e_1 \perp (e_2 \perp e_3) \equiv (e_1 \perp e_2) \perp e_3$;
- $\perp$ is not transitive: $(e_1 \perp e_2) \vee (e_2 \perp e_3)$ but $e_1 \neg \perp e_3$;
- $\#f$ is the neutral element for $\perp$: $e \perp \#f \equiv e$;
- $\#t$ is the absording element for $\perp$: $e \perp \#t \equiv \#t$.

*3) Concurrency:* $\mathcal{I}$ is the set of every couple of element of $\mathcal{EB}$ in concurrency. $e_1$ and $e_2$ are in concurrency if the occurrence of one is independent of the occurrence of the other. So, $e_1 \wr e_2$ iff $e_1$ and $e_2$ are neither in causality neither in exclusion.

$$e_1 \wr e_2 \equiv \neg((e_1 \perp e_2) \text{ or } (e_1 \prec e_2) \text{ or } (e_2 \prec e_1)) \tag{11}$$

- $\wr$ is commutative: $e_1 \wr e_5 \equiv e_5 \wr e_1$;
- $\wr$ is associative: $e_1 \wr (e_5 \wr e_7) \equiv (e_1 \wr e_5) \wr e_7$;
- $\wr$ is not transitive: $(e_1 \wr e_5) \vee (e_5 \wr e_2)$ but $e_1 \perp e_2$;
- $\#t$ is the neutral element for $\wr$: $e \wr \#t \equiv e$;
- $\#f$ is an absorbing element for $\wr$: $e \wr \#f \equiv \#f$.

*4) Process:* $\oplus$ aggregates events in one process. Two events $e_1$ and $e_2$ are in the same process if $e_1$ causes $e_2$ or if $e_1$ is

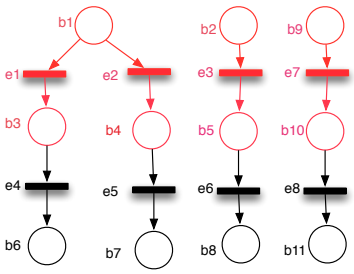


Figure 7. Concurrency.

concurrent with $e_2$:

$$e_1 \oplus e_2 \equiv (e_1 \prec e_2) \text{ or } (e_2 \prec e_1) \text{ or } (e_1 \wr e_2) \qquad (12)$$

This operator constitutes an abstraction that hides in a black box causalities and concurrencies. The meaning of this operator is similar to the linear connector $\oplus$ of MILL [18]. It allows to aggregates resources. But, in the context of unfolding, events or conditions are unique and then they cannot be counted. Thus, this operator is here idempotent.

The expression $e_1 \oplus e_2$ defines that $e_1$ and $e_2$ are in the same process.

Note that $(\oplus\ e_1\ e_2\ ...\ e_{n-1}\ e_n)$ will abbreviate $(e_1 \oplus\ e_2 \oplus e_3 \oplus ...e_{n-1} \oplus\ e_n)$

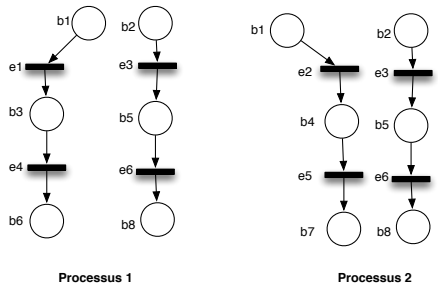


Figure 8. Process.

- $\oplus$ is commutative, associative, and transitive (definition of $\oplus$);
- Idempotency: $e \oplus e \equiv e$
- Neutral element: $e \oplus \#t \equiv e$
- Absorbing element: $e \oplus \#f \equiv \#f$
- $e \oplus \neg e \equiv \#f$

### C. Axioms

The following axioms stem directly from previous assumptions and definitions made upon the algebraic model:

**Axiom 1** (Distributivity of $\prec$)**.**

$$e \prec (e_1 \perp e_2) \equiv_{def} (e \prec e_1) \perp (e \prec e_2)$$

This first axiom constitutes the basis of our approach. As discussed in the Section II, on the contrary of $CCS$, $e$ is distributed onto two expressions, giving alternative processes.

**Axiom 2** (Definition of $\oplus$)**.**

$$e_1 \oplus e_2 \equiv_{def} (e_1 \prec e_2) \perp (e_2 \prec e_1) \perp (e_1 \wr e_2)$$

$\oplus$ aggregates two elements in a process. Two elements are in a process if they are concurrent or in a causality relation.

**Axiom 3** ($\prec$)**.**

$$e_1 \prec e_2 \equiv_{def} \neg e_1 \perp (e_1 \oplus e_2)$$

A causality can be expressed by two processes in exclusion: either $\neg e_1$: $e_1$ has not occurred either $e_1 \oplus e_2$: $e_1$ and $e_2$ within the same process.

**Axiom 4** (Duality between $\oplus$ and $\perp$)**.**

$$e_1 \oplus e_2 \equiv_{def} e_1 \neg\perp e_2 \qquad e_1 \neg\oplus e_2 \equiv_{def} e_1 \perp e_2$$

This axiom comes from the introduction of the operator $\neg$ discussed in the beginning of the Section IV. It expresses that $\mathcal{P}$ and $\mathcal{X}$ are complementary sets.

**Axiom 5** (Exclusion)**.**

$$e_1 \perp e_2 \equiv_{def} (\neg e_1 \oplus e_2) \perp (e_1 \oplus \neg e_2)$$

The fifth axiom expresses that a conflict can be considered as two processes in conflict.

### D. Distributivities

The distributivities over $\perp$ are used in the transformation of an expression in the canonical form (Section V). The other distributivities will be used in the reduction process.

*1) Distributivities over $\wr$:*

- $\prec$ is distributive over $\wr$:

$$e \prec (e_1 \wr e_2) \equiv (e \prec e_1) \wr (e \prec e_2)$$

- $\perp$ is distributive over $\wr$:

$$e \perp (e_1 \wr e_2) \equiv (e \perp e_1) \wr (e \perp e_2)$$

- $\oplus$ is distributive over $\wr$:

$$e \oplus (e_1 \wr e_2) \equiv (e \oplus e_1) \wr (e \oplus e_2)$$

*2) Distributivities over $\perp$:*

- $\prec$ is distributive over $\perp$ (Axiom 1):

$$e \prec (e_1 \perp e_2) \equiv (e \prec e_1) \perp (e \prec e_2)$$

- $\wr$ is distributive over $\perp$:

$$e \wr (e_1 \perp e_2) \equiv (e \wr e_1) \perp (e \wr e_2)$$

- $\oplus$ is distributive over $\perp$:

$$e \oplus (e_1 \perp e_2) \equiv (e \oplus e_1) \perp (e \oplus e_2)$$

*3) Distributivities over $\oplus$:*

- $\perp$ is distributive over $\oplus$:

$$e \perp (e_1 \oplus e_2) \equiv (e \oplus e_1) \perp (e \oplus e_2)$$

- $\wr$ is distributive over $\oplus$:

$$e \wr (e_1 \oplus e_2) \equiv (e \oplus e_1) \wr (e \oplus e_2)$$

### E. Derivation Rules

This section gives a set of rules, which transform branching processes toward a canonical form. These transformations preserve conflicts whereas $\prec$ and $\wr$ are transformed in $\oplus$.

Let us note $b$ a condition, $e$ an event and $E$ a well formed formula on the algebra. These rules allow to reduct process:

1) Modus Ponens:

$$\frac{\vdash \oplus\ b\ ... \quad \vdash \oplus\ b\ ... \prec e}{\vdash e}\ MP_1$$

$$\frac{\vdash e \quad \vdash e \prec \oplus\ b\ ...}{\vdash \oplus\ e\ b\ ...}\ MP_2$$

Where $\oplus\ b\ ...$ stands for the general form for $(\oplus\ b_1\ b_2\ ...\ b_n)$. $MP_1$ expresses that the set of conditions $\oplus\ b\ ...$ are consumed by the causality, whereas in $MP_2$, $e$ stays in the conclusion.

2) Dual form:

$$\frac{\vdash \neg e_1 \quad \vdash e_1 \prec e_2}{\vdash \neg e_1 \oplus \neg e_2}\ MP'$$

3) Simplification:

$$\frac{\vdash \neg e_1 \oplus E}{\vdash E}\ S_1$$

$$\frac{\vdash \oplus\ b\ ...\ E}{\vdash E}\ S_2$$

Those rules are applied, *in fine*, to clear not pertinent informations in the process. $S_1$ rule is applied, to clear the negations, whereas $S_2$ is applied to clear the conditions, which have not been consumed.

4) Reduction of $\wr$:

$$\frac{\vdash e_1 \wr e_2}{\vdash e_1 \oplus e_2}\ Par$$

This rule corresponds to the definition of $\oplus$
These rules have been defined to lead to a canonic form.

## V. Canonic Form and Conflict Equivalence

A *canonic form* is a relation expressed on elements of $\mathcal{EB}$ and with the operators $\oplus$ and $\perp$ ordered by an alphanumeric sort on the name of its symbol. This definition of the canonic form allows to define an equivalence called a "conflict equivalence".

**Theorem 1** (Canonical form)**.** *Let us consider an unfolding $U$, this form can be reduced in the following form:*

$$U = \ (\perp\ P_1\ P_2\ ...\ P_n), \text{where} \quad P_i = (\oplus\ e_{i_1}... \ e_{i_n})$$

*This form is canonic and exhibits every processes $P_i$ of the unfolding.*

*Proof.* In an unfolding every causality ($\prec$) and every partial order ($\wr$) can be reduced in $\oplus$ by deduction rules Modus Ponens ($MP, MP_1, MP_2$), Simplification rule ($S$) and $Par$ (see Section IV-E).

Moreover, $\oplus$ and $\perp$ are mutually distributive, so $\perp$ can be factorized in every sub-formula to reach the higher level of the formula. In fine, an alphanumeric sort on symbols of the processes can be applied to assure the unicity of the form. □

This canonic form preserves conflicts, let us now define a *conflict equivalence*:

**Definition 5** (Conflict Equivalence)**.** *Let us $U_1, U_2$ unfoldings of Petri nets:*

$$U_1 \approx_{conf} U_2 \qquad \textit{iff they have the same canonic form.}$$

**Remark 2.** *A process is an aggregate set of events, where $\prec$ and $\wr$ are hidden. This equivalence is lower than a trace equivalence: each process $P_i$ is an abstraction of a set of traces.*

### A. Theorems

The properties of operators (definitions, axioms and distributivites) allow to define theorems, which are congruences.

**Theorem 2** (Conflict)**.**

$$e_1 \prec (e_2 \perp e_3) \equiv (e_1 \prec (e_2 \oplus \neg e_3)) \perp (e_1 \prec (\neg e_2 \oplus e_3))$$

*Proof.*

$$\begin{aligned} e_1 \prec (e_2 \perp e_3) \quad &\equiv_{Ax_5} \quad e_1 \prec ((e_2 \oplus \neg e_3) \perp (\neg e_2 \oplus e_3) \\ &\equiv_{dist} \quad (e_1 \prec (e_2 \oplus \neg e_3)) \perp (e_1 \prec (\neg e_2 \oplus e_3)) \end{aligned}$$

□

This theorem expresses how to develop a conflict and the following theorem allows to reduce processes:

**Theorem 3** (Absorption)**.** *Let $E, F$ some processes:*

$$E \perp (E \oplus F) \equiv E \oplus F$$

*Proof.*

$$\begin{array}{rcl} E \perp (E \oplus F) & \equiv & (E \oplus \#t) \perp (E \oplus F) \\ & \equiv_{Neutral} & E \oplus (\#t \perp F) \\ & \equiv & E \oplus F \end{array}$$

□

### B. Chain of conflicts

This section presents a theorem that computes the branching process in canonic form of a chain of conflict illustrated in Figure 9.

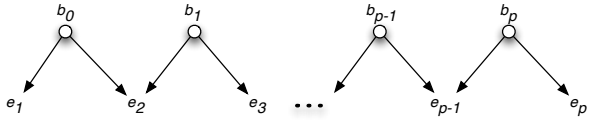


Figure 9. Chain of conflicts.

The axiomatic representation of the unfolding is:

$$U = ((\oplus\ b_0\ b_1\ ...\ (b_0 \prec (e_1 \perp e_2))(b_1 \prec (e_2 \perp e_3))...)$$

After some steps of reduction $(MP + S)$:

$$U = (e_1 \perp e_2 \perp ... \perp\ e_p)$$

Let us note:

- $l^1 = (e_1, e_2, ...e_n)$, $l^2 = (e_2, ...e_n)$
- $l_i$ the $i^{th}$ element of a list $l$.
- If $e_i$ is an element of the list $l$, let us note $indice(e_i)$ the position of $e_i$ in $l$.

**Remark 3.** *In the list of event constituting a chain of conflict ($l = (e_1, e_2, ...e_n)$), for every event $e_i$, the next (resp. previous) event in the same branching process is $e_{i+2}$ or $e_{i+3}$ (resp. $e_{i-2}$ or $e_{i-3}$)*

The next definition defines two processes $U_n$ and $V_n$, which are aggregation of events, where the possible successor of an event $e_i$ is either $l_{(indice(e_i)+2)}$ either $l_{(indice(e_i)+3)}$.

**Definition 6.** *Let us consider that $n <= p$,*

$$\left\{\begin{array}{l} U_0 = e_1 \\ U_n^1 = l_{n+2}^1 \oplus U_{n+2}^2 \\ U_n^2 = l_{n+3}^1 \oplus U_{n+3}^2 \\ U_n = U_n^1 \ \oplus U_n^2 \end{array}\right. \qquad \left\{\begin{array}{l} V_0 = e_2 \\ V_n^1 = l_{n+2}^2 \oplus V_{n+2}^2 \\ V_n^2 = l_{n+3}^2 \oplus V_{n+3}^2 \\ V_n = V_n^1 \ \oplus V_n^2 \end{array}\right.$$

*$U_n$: processes beginning by $e_1$ $V_n$: processes beginning by $e_2$ where $p$ is the index of the last event implied in the chain of conflict*

**Theorem 4.** *The canonic form of a chain of conflict $C$ is $U_n \oplus V_n$:*

$$(e_1 \perp\ e_2 \perp\ ...\perp\ e_p) \equiv U_n \oplus V_n$$

*Proof. Correctness:* let us consider an incorrect process $q \in L_p$:

$$q = (\oplus\ e_{q_1}\ e_{q_2}\ ...\ e_{q_p})$$

An incorrect process contains two event in conflict. Thus, this incorrectness implies the existence of two events in $q$ such as $e_{q_i} \perp e_{q_{i+1}}$ and $e_{q_i}, e_{q_{i+1}}$ corresponding to two successive events of $l$. This is in contradiction with the definition of the functions $(U_n^1, U_n^2, V_n^1, V_n^2)$ for which events are added with either $l_{n+2}$ either $l_{n+3}$. For a correct process, indices cannot be consecutive.

*Completeness:* let us consider a valid process:

$$q = (\oplus\ e_{q_1}\ e_{q_2}\ ...\ e_{q_p})$$

which is not included in $L_p$. $\forall e \in q$, if $q$ is valid then $\forall (e_i, e_j) \in q, \neg(e_i \perp e_j)$, so it implies that $e_i$ and $e_j$ are not successive in $l$ and every enabled event is in $q$. Moreover, as $q$ is not included in $L_p$, thus, it exists at least one couple $(e_{q_i}, e_{q_j})$, which does not correspond to the construction defined by the functions $(U_n^1, U_n^2, V_n^1, V_n^2)$, which define the possible successor of an event. This means that $indice(e_{q_j}) > indice(e_{q_i} + 3)$.

For every $n = indice(e_{q_j}) - indice(e_{q_i})$ greater than 3, let us note $i_2 = indice(e_{q_i}) + 2$ the event $e_{q_{i_2}}$ is a possible event, which is not in $q$ (contradiction). □

## VI. Examples

Examples VI-A and VI-B illustrate conflit equivalence, whereas the example VI-C contains reset arcs.

### A. Example 1

Figure 10 gives a Petri net, which represents a chain of conflicts and its unfolding.

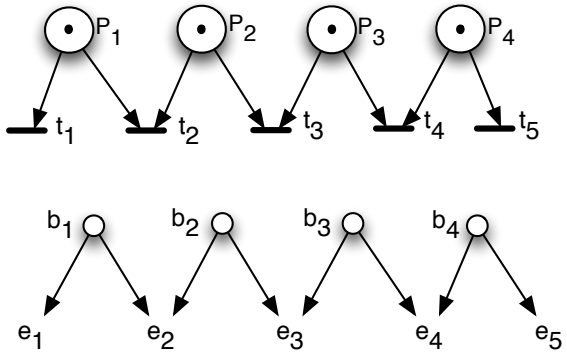


Figure 10. PN and unfolding of a chain of conflicts.

The unfolding gives a table of binary relations on events (see Section IV), which is represented by the following algebraic expression $U_2$:

$$U_1 = (\oplus\ b_1\ b_2\ b_3\ b_4\ b_5\ (b_1 \prec (e_1 \perp e_2))\ (b_2 \prec (e_2 \perp e_3))\ ...)$$

After some steps of reduction $(MP + S)$, $U_1$ becomes:

$$(e_1 \perp\ e_2 \perp\ e_3 \perp\ e_4 \perp\ e_5) \tag{13}$$

Theorem 4 allows to compute from (13) its following canonic form:

$$(\perp (\oplus\ e_1\ e_3\ e_5)(\oplus\ e_1\ e_4\ )(\oplus\ e_2\ e_4)(\oplus\ e_2\ e_5))$$

### *B. Example 2*

Let us consider the following Unfolding of Figure 11. The

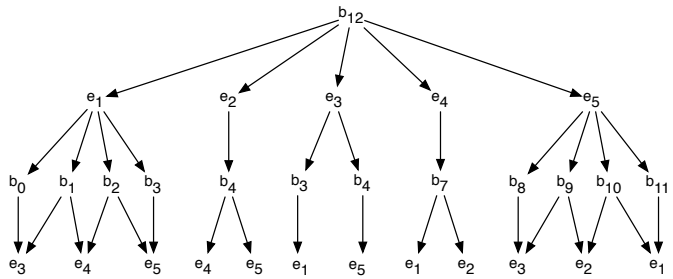


Figure 11. $U_2$.

table has been computed and the set of binaries relations between events leads to the following algebraic expression $U_2$:

$$\begin{aligned} U_2 \quad = (\oplus\ b_{12} \quad & (b_{12} \prec (e_1 \perp\ e_2 \perp\ e_3 \perp\ e_4 \perp\ e_5)) \\ & (e_1 \prec (\oplus\ b_0\ b_1\ b_2\ b_3))(e_2 \prec b_4)(e_3 \prec (\oplus b_5\ b_6)) \\ & (e_5 \prec (\oplus b_8\ b_9\ b_{10}\ b_{11}))((\oplus\ b_0\ b_1) \prec e_3) \\ & ((\oplus\ b_1\ b_2) \prec e_4)\ (e_4 \prec b_7)\ ((\oplus\ b_2\ b_3) \prec e_5) \\ & (b_4 \prec (\perp e_4\ e_5))(b_5 \prec e_1)\ (b_6 \prec e_5) \\ & (b_7 \prec (\perp e_1\ e_2))\ ((\oplus\ b_8\ b_9) \prec e_3) \\ & ((\oplus\ b_9\ b_{10}) \prec e_2)\ ((\oplus\ b_{10}\ b_{11}) \prec e_1)) \end{aligned} \tag{14}$$

Let us note $P$ the aggregation of the five first lines of the previous Equation (14) becomes:

$$U_2 \quad = (\oplus\ b_{12} \quad (b_{12} \prec (\perp\ e_1\ e_2\ e_3\ e_4\ e_5))\ P \tag{15}$$

Rules $MP1$, $MP_2$ and theorem 1 reduce (15) in:

$$\begin{aligned} U_2 \quad = (\perp\ & (\oplus\ e_1\ P)\ (\oplus\ e_2\ P)\ (\oplus\ e_3\ P) \\ & (\oplus\ e_4\ P)\ (\oplus\ e_5\ P)\ ) \end{aligned}$$

Distributivity of $perp$:

$$\begin{aligned} U_2 \quad = (\oplus \quad & (\perp (\oplus\ e_1\ b_0\ b_1\ b_2\ b_3)(\oplus\ e_2\ b_4)(\oplus\ e_3\ b_5\ b_6) \\ & (\oplus\ e_4\ b_7)(\oplus\ e_5\ b_8\ b_9\ b_{10}\ b_{11}))\ ((\oplus\ b_0\ b_1) \prec e_3) \\ & ((\oplus\ b_1\ b_2) \prec e_4)((\oplus\ b_2\ b_3) \prec e_5)\ (b_4 \prec (\perp e_4\ e_5)) \\ & (b_5 \prec e_1)\ (b_6 \prec e_5)(b_7 \prec (\perp e_1\ e_2)) \\ & ((\oplus\ b_8\ b_9) \prec e_3)\ ((\oplus\ b_9\ b_{10}) \prec e_2) \\ & ((\oplus\ b_{10}\ b_{11}) \prec e_1)) \end{aligned}$$

Distributivity of $\perp$ and $MP_1$:

$$\begin{aligned} U_2 \quad = (\perp \quad & (\oplus\ e_1\ e_3\ e_5\ b_1\ b_2)(\oplus\ e_1\ e_4\ b_0\ b_3)(\oplus\ e_2\ e_4) \\ & (\oplus\ e_2\ e_5)\ (\oplus\ e_3\ e_1)(\oplus\ e_3\ e_5)(\oplus\ e_4\ e_1) \\ & (\oplus\ e_4\ e_2)(\oplus\ e_5\ e_1\ e_3\ b_9\ b_{10})\ (\oplus\ e_5\ e_2\ b_8\ b_{11})) \end{aligned}$$

Theorem 2 : absorption of $(\oplus\ e_3\ e_1)$ and $(\oplus\ e_3\ e_5)$ in $(\oplus\ e_1\ e_3\ e_5\ b_1\ b_2)$, idempotency of $\perp$:

$$\begin{aligned} U_2 \quad = (\perp \quad & (\oplus\ e_1\ e_3\ e_5\ b_1\ b_2)(\oplus\ e_1\ e_4\ b_0\ b_3)(\oplus\ e_2\ e_4) \\ & (\oplus\ e_2\ e_5)\ (\oplus\ e_4\ e_1)(\oplus\ e_5\ e_1\ e_3\ b_9\ b_{10}) \\ & (\oplus\ e_5\ e_2\ b_8\ b_{11})) \end{aligned}$$

Rules of simplification $S_1$ and $S_2$ and theorem 2:

$$U_2 = (\perp (\oplus\ e_1\ e_3\ e_5)(\oplus\ e_1\ e_4)(\oplus\ e_2\ e_4)(\oplus\ e_2\ e_5))$$

The two unfoldings of examples 1 and 2 have the same canonic form, they are *conflict-equivalent*: $U_1 \approx_{conf} U_2$

*1) Reasoning about processes:* Let us consider all the process $p$ of $U_2 : (\oplus\ e_1\ e_3\ e_5), (\oplus\ e_1\ e_4), ...$

- $\forall p \in U_2$ whenever $e_3$ is present, $e_1$ is present.
- $\forall p \in U_2, \neg e_3 \perp (e_1 \oplus\ e_3 \oplus e_5)$
  This is the algebraic definition of $\prec$. Finally, from this chain of conflicts, the following causality can be deduced:

$$e_3 \prec (e_1 \oplus e_5) \tag{16}$$

- A similar reasoning can be made:

$$\forall p \in U_2, \neg(e_1 \oplus e_5) \perp (e_1 \oplus\ e_3 \oplus e_5)$$

  This is the algebraic definition of:

$$(e_1 \oplus e_5) \prec e_3 \tag{17}$$

  Equations (16) and (17) express that there is a *strong link* between $e_3$ and the process $(e_1 \oplus e_5)$ but $\prec$ is no well suited to encompass this relation. These two processes are like "intricated".
- In the same manner:

$$\begin{aligned} & \neg e_2 \perp (e_2 \oplus e_4) \perp (e_2 \oplus e_5) \\ \equiv_{dist}\ & \neg e_2 \perp (e_2 \oplus (e_4 \perp e_5)) \\ \equiv_{def}\ & e_2 \prec (e_4 \perp e_5) \end{aligned} \tag{18}$$

  $e_2$ leads to a conflict

$$\begin{aligned} & \neg e_1 \perp ((\oplus e_1 e_3 e_5) \perp (e_1 \oplus e_4) \\ \equiv_{dist}\ & \neg e_1 \perp (e_1 \oplus ((e_3 \oplus e_5) \perp e_4)) \\ \equiv_{def}\ & e_1 \prec ((e_3\ \oplus\ e_5) \perp e_4) \end{aligned} \tag{19}$$

Equations (18) and (19) show that $e_1$ and $e_2$ transform the chain of conflict in a unique conflict. New relations between events or processes can be introduced:

- Alliance relation: $e_1, e_3$ and $e_5$ are in "an alliance relation". Every event of this set is enforced by the occurrence of the other events: $e_1 \oplus e_3$ enforces $e_5$, $e_1 \oplus e_5$ enforces $e_3$ and $e_3 \oplus e_5$ enforces $e_1$.
- Intrication: the occurrence of $e_3$ forces $e_1 \oplus e_5$ and reciprocally $e_1 \oplus e_5$ forces $e_3$.
- Resolving conflicts (liberation):
  - $e_1$ resolves 3 conflicts on 4 (as $e_2$, $e_4$ and $e_5$)
  - $e_3$ resolves every conflicts.

Semantically, $e_3$ can be identified as an important event in the chain. Moreover, $(\oplus e_1\ e_3\ e_5)$ is a process aggregated with "associated events". This chain of conflict can be seen as two causalities in conflicts: $(e_1 \prec (e_4 \perp (e_3 \oplus e_5)))\ \perp\ (e_2 \prec (e_4 \perp e_5))$

### C. Example 3 (Cash dispenser)

Let us consider a cash dispenser illustrated in Figure 12. The user has three tries (3 tokens are generated in place WaitEnterCode) to enter a valid code ($OKcode$), then he can get $Cash$ or can $Consult$ its account. In this example, a reset arc from $OKCode$ allow to clear the tokens that have not be consumed (for example when the user has entered a valid code at its first or second try) and two reset arcs have been added from $getConsult$ and $getCash$ to clear $ReadyToConsult$ or $ReadyToGetCash$.

It could be useful to prove that if the events $GetCash$ implies that $Okcode$ belongs to the same process.

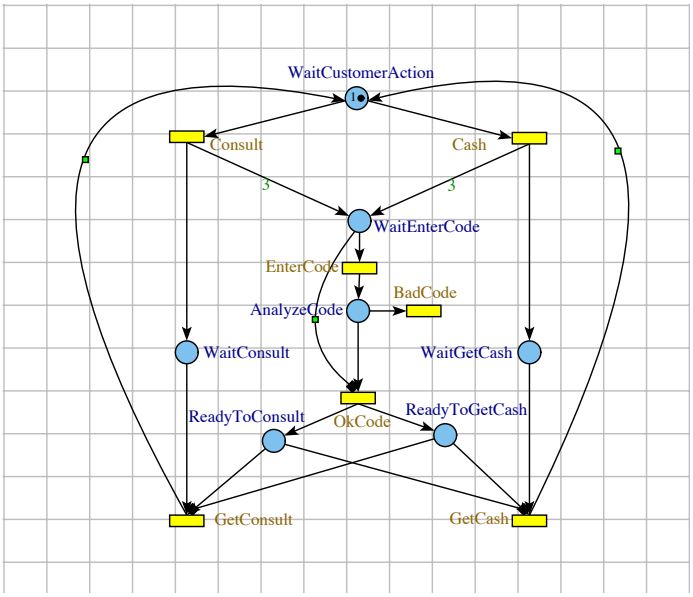


Figure 12. Cash dispenser.

The unfolding of cash dispenser is given in Figure 13. A combinatory inflation of the net is caused by to the reset arcs and by the transitions, $Consult$ and $Cash$, which produces 3 tokens each.

The reset arcs introduces for each events $e_9$, $e_{10}$, $e_{11}$, $e_{18}$, $e_{19}$, and $e_{20}$ (events relative the transition $OKcode$) two arcs, which consumes adding conditions. The translation of reset arcs have been defined manually and is not yet implemented in reduction rules. The computing of the canonical form of the processes is following expression:

$U3 = (\perp (\oplus\ Consult\ EnterCode\ OKcode\ GetConsult)$
$(\oplus\ Consult\ EnterCode\ BadCode\ OKcode\ GetConsult)$
$(\oplus\ Consult\ EnterCode\ BadCode\ \ BadCode\ OKcode\ GetConsult)$
$(\oplus\ Consult\ EnterCode\ BadCode\ BadCode\ BadCode)$
$(\oplus\ Cash\ EnterCode\ OKcode\ Getcash)$
$(\oplus\ Cash\ EnterCode\ BadCode\ OKcode\ Getcash)$
$(\oplus\ Cash\ EnterCode\ BadCode\ BadCode\ OKcode\ Getcash)$
$(\oplus\ Cash\ EnterCode\ BadCode\ BadCode\ BadCode)$

This expression formally proves that if $GetCash$ is in a process then $OkCode$ belongs to the same process.

## VII. Implementation aspects

A program [19] has been developed. It takes Petri Nets as inputs Romeo [20] unfolds and computes the canonical form. This program has been written in Lisp. The algebraic definitions and the reduction rules has been described with *redex*, a formal package introduced in [11].

### A. Syntax of the language

The redex package allows to implement the syntactic rules of the language with an abstract and conceive way:

```
; Nodes                                                    1
   [bool t f]                                              2
   [n variable bool b e (¬ ⊕ n)]                           3
   [e variable (¬ e)]                                      4
   [b variable (¬ b)]                                      5
; n−ary or binary operators                                6
   [on ⊕ ⊥ ≀]                                              7
   [o2 ≺]                                                  8
; Process                                                  9
   [P variable (⊕ Q ... )]                                 10
   [Q variable P n]                                        11
   [C−P (⊕ C−P P) (⊕ P C−P) hole]                          12
;  Conflicts                                               13
   [X variable (⊥ Y ... )]                                 14
   [Y variable X n ]                                       15
   [C−X (⊥ C−X P) (⊥ P C−X) hole]                          16
;  Expression                                              17
   [E variable (on F ...)                                  18
                    (b o2 e) (P o2 X)]                     19
   [F variable E P ]                                       20
   [C−E (on C−E E) (on E C−E)                              21
          (E o2 C−E) (C−E o2 E) hole]                      22
```

- The lines 2 to 5 define the basics nodes, which are boolean, $b$ conditions and $e$ the events. The term *variable* in lines 3 to 5 allows to use in the language every symbols denoted as $n_i$, $b_i$ or $e_i$. These symbols are the terminal symbols of the alphabet.
- The lines 7 and 8 group the n-ary and the binary operators.
- Lines 10 to 12 define the process. A process $P$ is constitued with $\oplus$ operator on $Q$, where $Q$ is defined as a node $n$ or a process $P$. Every non terminal symbol $P_i$ is a process.
- Lines 14 to 16 define conflicts in a similar way.
- Finally, lines 18 to 22 define expressions that are built from conflicts, process and causality.
- For every term: Process, Conflicts and Expression, contexts are defined. The contexts capture prefixes

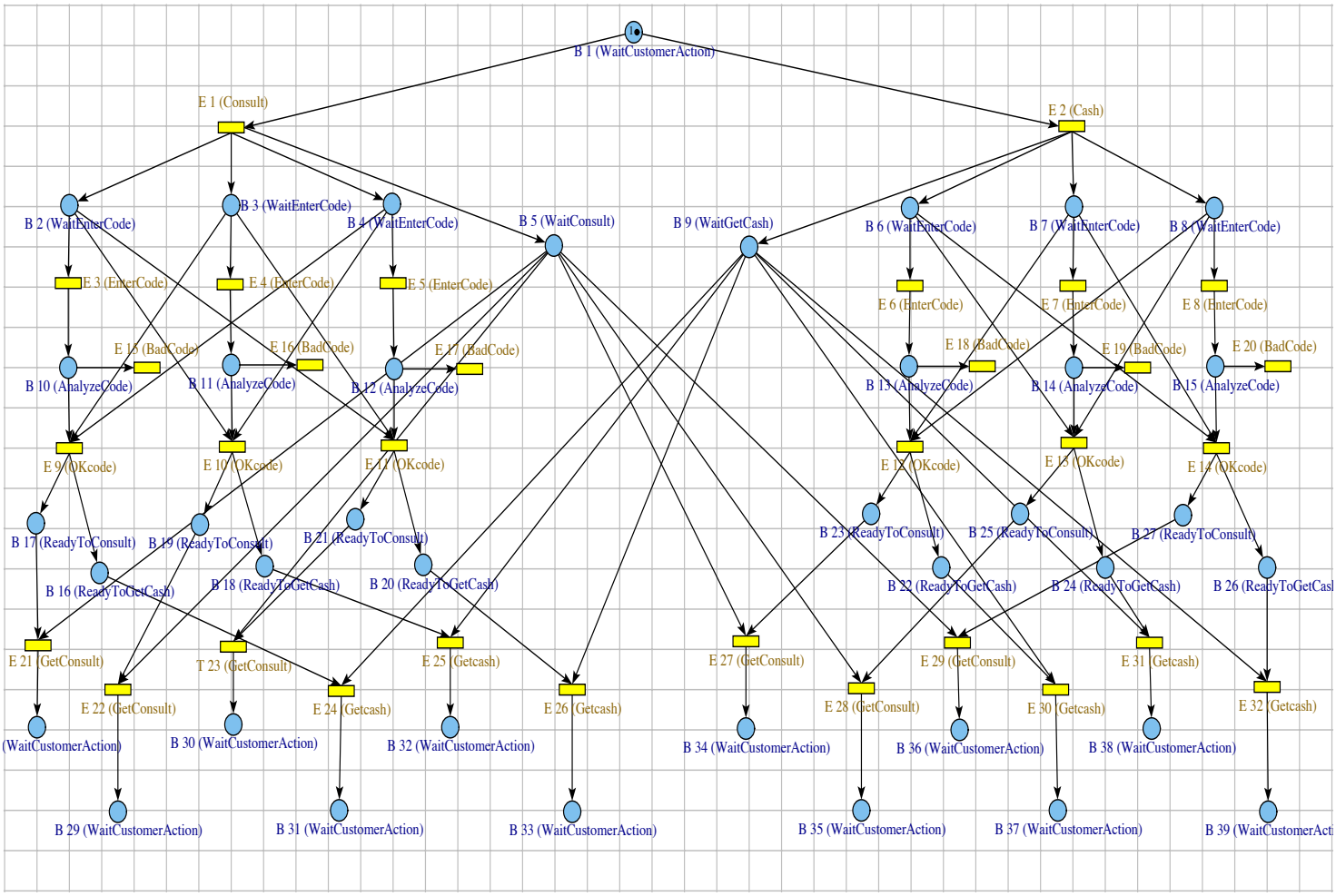


Figure 13. Unfolding of Cash dispenser.

and suffixes of an expression and put them into a hole.

### B. Reductions rules

Definitions have been implemented as reduction rules:

```
(-->  (in-hole C-P
        (⊕ Q_1 ... f Q_2 ...))
      (in-hole C-P f)  "A⊕" )
(-->  (in-hole C-E
        (⊕ Q_1 ... e_1 Q_2 ... (¬ e_1) Q_4 ...))
      (in-hole C-E f)  "F⊕" )
```

The particularities of this syntax are:

- $Q_i$... is equivalent to the regular expression $Q_i^*$, which represents an ordered list of symbol $Q_i$, which is eventually empty, finite or infinite.
- The contexts C-P or C-E allows to capture every sub-expression with every prefix and suffixe.

The first rule, labelled $A_\oplus$, illustrates that $f$ is an absorbing element. In this rule, C-P captures the context of a Process $P$ and put into a hole. This reduction rule expresses that every sub-expression of the type $(\oplus Q_1 ... f Q_2 ...)$, which can be reduced to the node $f$. This rule is named and thus, its use can be traced in a future proof.

The second rule $F_\oplus$ states the property defined in Section IV-B4 : $e_1 \oplus \neg e_1 \equiv \#f$. This reduction rules defines that every expression (for every context C-E) containing $e_1$ and $\neg e_1$ in an $\oplus$ operator can be reduced to $f$.

### C. Theorems

This section describes the implementation and the coding of theorems.

*1) Theorem 4:* Theorem 4 has been stated from definition 6, which corresponds to the following statements:

```
(define (U1n n 1)
  (if (>= (- (maxi 1) n) 2)
      (cons (list-ref 1 (+ n 2))
            (Rn (+ n 2) 1)) empty))

(define (V2n n 1)
  (if (>= (- (maxi 1) n) 3)
      (cons (list-ref 1 (+ n 3))
            (Rn (+ n 3) 1)) empty))
```

Finally, the implementation is coded like the union of the previous definitions:

```
(define (Rn n l)
  (if (>= (- (maxi l) n) 1)
      (Union (U1n n l) (U2n n l))
      empty))
```

Note that the implementation of the definitions and the theorems are closed to their formal expression.

*2) Theoreme 3:* $E \perp (E \oplus F) \equiv (E \oplus F)$ has been implemented has a reduction rule:

```
(-->  (in-hole C-E
         (⊥ E_1 ... E E_2 ...
              (⊕ E E_3 ...) E_4 ...))
      (in-hole C-E
            (⊥ E_1 ... E_2 ...
                 (⊕ E E_3 ... ) E_4 ...)) "T3")
```

This code means that if $E$ is in a "$\perp$ expression:" ($\perp$ $E_1$... $EE_2$...), then if a sub expression in $\oplus$ contains $E$, then $E$ can be suppressed of the "$\perp$ expression" for any context.

## VIII. Conclusion and future work

This work is a first attempt to present an axiomatic framework to the analyze of the processes issued of an unfolding. From a set of axioms, distributivities, and derivation rules, theorems have been established and a reduction process can lead to a canonic form. The unfolding process, definitions, theorems, and reduction rules have been coded in LISP[21] with a package named PLT/Redex[11][22]. This canonic form assets an equivalence conflicts ($\equiv_{conf}$) between unfoldings and then Petri nets.

Several perspectives are into progress. First, new theorems have to be established allowing to speed up the procedure of canonic reduction and to extend extraction of knowledge on relationship between events. Different kinds of relationship between events can be defined and formalized: Alliance relation, Intrication, etc. Moreover, as already outlined in the examples, algebraic reasoning can raise semantic informations about events from the canonic form. Another perspective is to extend the approach to Petri nets with inhibitor or drain arcs.

## References


[1] d. Delfieu, M. Comlan, and M. Sogbohossou, "Algebraic analysis of branching processes," in *Sixth International Conference on Advances in System Testing and Validation Lifecycle*, 2014, pp. 21–27, best paper award.

[2] J. Esparza and K. Heljanko, "Unfoldings - a partial-order approach to model checking," *EATCS Monographs in Theoretical Computer Science*, 2008.

[3] Engelfriet and Joost, "Branching processes of petri nets," *Acta Informatica*, vol. 28, no. 6, pp. 575–591, 1991.

[4] J. Esparza, S. Römer, and W. Vogler, *An Improvement of McMillan's Unfolding Algorithm*. Mit Press, 1996.

[5] McMillan and Kenneth, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *Computer Aided Verification*. Springer, 1993, pp. 164–177.

[6] C. A. R. Hoare, *Communicating sequential processes*. Prentice-hall Englewood Cliffs, 1985, vol. 178.

[7] R. Milner, *Communication and concurrency*. Prentice-hall Englewood Cliffs, 1989.

[8] C. A. Petri, "Communication with automata," *PhD thesis, Institut fuer Instrumentelle Mathematik*, 1962.

[9] R. Glabbeek and F. Vaandrager, "Petri net models for algebraic theories of concurrency," in *PARLE Parallel Architectures and Languages Europe*, ser. Lecture Notes in Computer Science, J. Bakker, A. Nijman, and P. Treleaven, Eds. Springer Berlin Heidelberg, 1987, vol. 259, pp. 224–242.

[10] E. Best, R. Devillers, and M. Koutny, "The box algebra=petri nets+process expressions," *Information and Computation*, vol. 178, no. 1, pp. 44 – 100, 2002.

[11] M. Felleisen, R. Findler, and M. Flatt, *Semantics Engineering With PLT Redex*. Mit Press, 2009.

[12] M. Nielsen, G. Plotkin, and G. Winskel, "Petri nets, event structures and domains," in *T. Theor. Comp. Sci.*, vol. 13(1), 1981, pp. 89–118.

[13] J. Baeten, J. Bergstra, and J. Klop, "An operational semantics for process algebra," in *CWI Report CSR8522*, 1985.

[14] G. Boudol and I. Castellani, "On the semantics of concurrency: partial orders and transitions systems," in *Rapports de Recherche No 550, INRIA, Centre Sophia Antipolis*, 1986.

[15] V. Glaabeek and F. Vaandrager, "Petri nets for algebraic theories of concurrency," in *CWI Report SC-R87*, 1987.

[16] C. Dufourd, P. Jančar, and Ph. Schnoebelen, in *Proceedings of the 26th ICALP'99*, ser. Lecture Notes in Computer Science, J. Wiedermann, P. van Emde Boas, and M. Nielsen, Eds., vol. 1644. Prague, Czech Republic: Springer, Jul. 1999, pp. 301–310.

[17] T. Chatain and C. Jard, "Complete finite prefixes of symbolic unfoldings of safe time petri nets," in *Petri Nets and Other Models of Concurrency - ICATPN 2006*, ser. Lecture Notes in Computer Science, S. Donatelli and P. Thiagarajan, Eds. Springer Berlin Heidelberg, 2006, vol. 4024, pp. 125–145. [Online]. Available: http://dx.doi.org/10.1007/11767589

[18] J.-Y. Girard, "Linear logic," *Theoretical computer science*, vol. 50, no. 1, pp. 1–101, 1987.

[19] D. Delfieu and M. Comlan, "Penelope," http://penelope.rts-software.org/svn, Oct. 2013, tools for editing, unfolding and and to obtain canonical form for Petri Nets.

[20] G. Gardey, D. Lime, M. Magnin *et al.*, "Romeo: A tool for analyzing time petri nets," in *Computer Aided Verification*. Springer Berlin Heidelberg, 2005, pp. 418–423.

[21] G. L. Steele, *Common LISP: the language*. Digital press, 1990.

[22] D. Delfieu and S. Mdssu, "An algebra for branching processes," in *Control, Decision and Information Technologies (CoDIT), 2013 International Conference on*, May 2013, pp. 625–634.